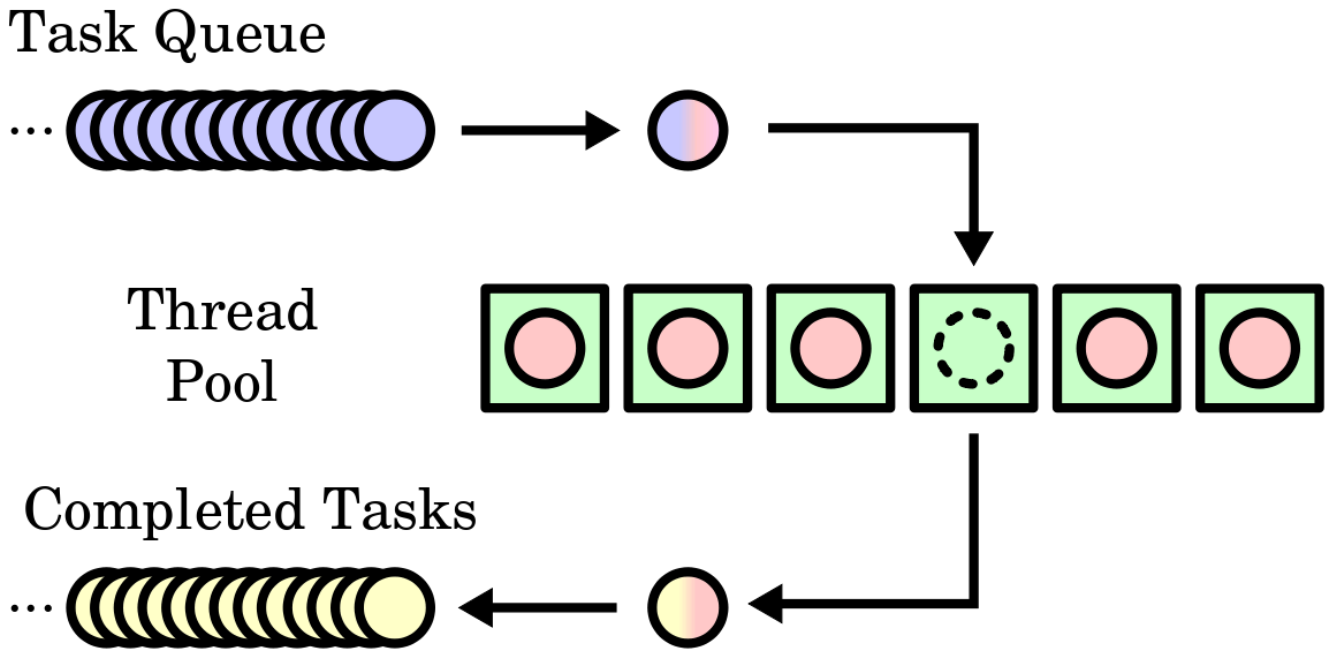
Конспект лекционного занятия от 25.10.2024

Синхронизация потоков

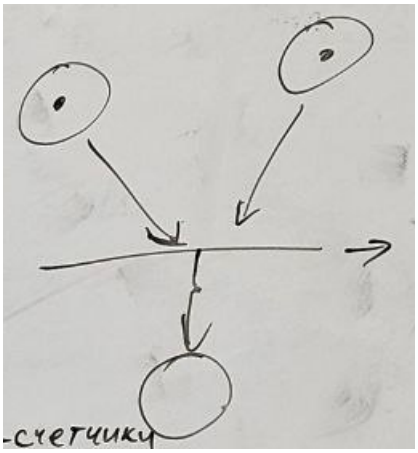
Пул потоков (Thread Pool) — это фиксированный набор потоков, одновременно выполняющих независимые друг от друга задачи, помещенные в некоторый



Основная задача синхронизации потоков – управление последовательностью выполнения потоков во времени. Взаимное исключение (mutual exclusion) для предотвращения конфликтов доступа.

Средства реализации:

- **Mutex** (мьютекс):
  - Используется для блокировки ресурсов.
  - Взаимное исключение достигается блокировкой сигнала.
- Сети Петри (IEEE стандарт):
  - Применяются для моделирования процессов.
  - Примеры:
    - POSIX (pthread библиотеки).
    - Спин-блокировки (spin-lock).



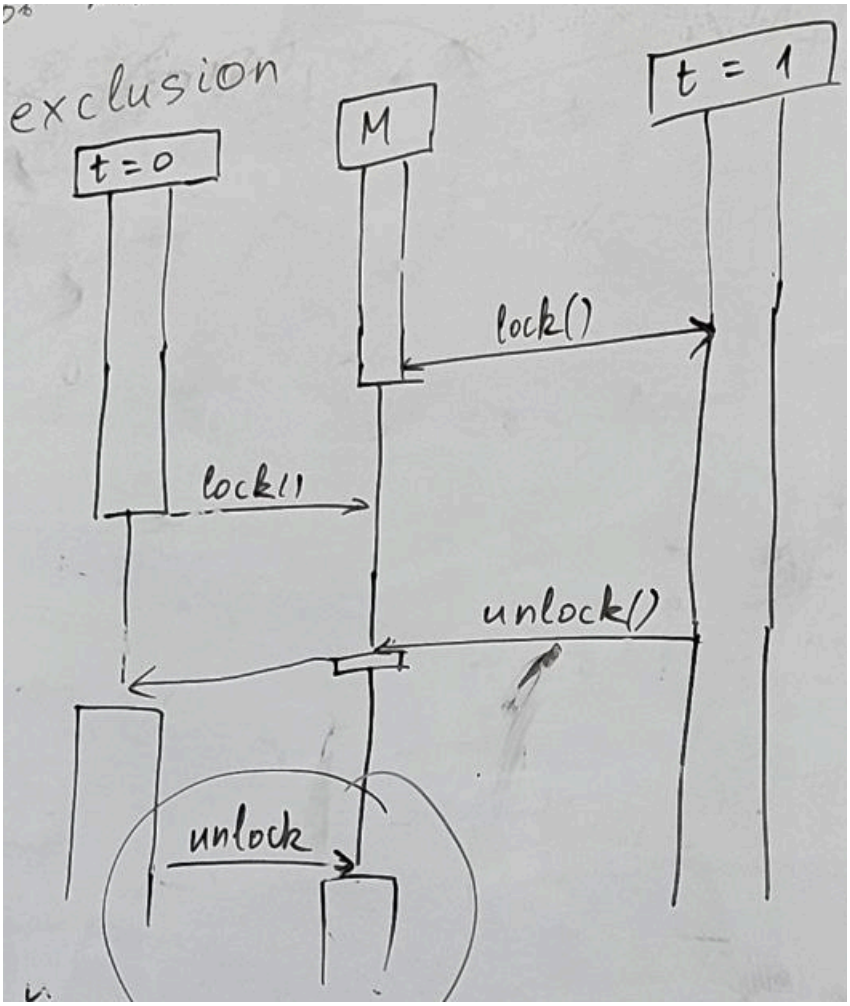
Мьютекс (mutex - mutuallyexclusive), который также называют защелкой - это механизм изоляции, используемый программами для синхронизации доступа нескольких потоков к совместно используемым ресурсам.

Мьютексы в языке C++

Стандарт C++17 языка программирования **C++** определяет 6 различных классов мьютексов:

- `mutex` — мьютекс без контроля повторного захвата тем же потоком[1];
- `recursive_mutex` — повторные захваты тем же потоком допустимы, ведётся подсчёт таких захватов[2];
- `timed_mutex` — нет контроля повторного захвата тем же потоком, имеет дополнительные методы захвата мьютекса с возвратом значения `false` в случае истечения тайм-аута или по достижении указанного времени[3];
- `recursive_timed_mutex` — повторные захваты тем же потоком допустимы, ведётся подсчёт таких захватов, имеет дополнительные методы захвата мьютекса с возвратом кода ошибки по истечении тайм-аута или по достижении указанного времени[4];
- `shared_mutex` — разделяемый мьютекс[5];
- `shared_timed_mutex` — разделяемый мьютекс, имеет дополнительные методы захвата мьютекса с возвратом кода ошибки по истечении тайм-аута или по достижении указанного времени[6].

Библиотека Boost дополнительно обеспечивает именованные и межпроцессные мьютексы, а также разделяемые мьютексы, которые позволяют захватывать мьютекс для совместного владения несколькими потоками только для чтения данных с запретом на эксклюзивную запись на время захвата блокировки, что по сути представляет собой механизм блокировок чтения и записи25(https://ru.wikipedia.org/wiki/%D0%9C%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81#cite_note-25).



Пример использования мьютексов:

Листинг 4.1

```
#pragma omp critical
{
    // Критическая секция
    sum += 1;

    while (ресурс занят) {
        // Ожидание освобождения ресурса
        вытесняться();
    }

    if (i < 6000 && ресурс занят) {
        вытесняться();
    } else if (i == 6000) {
        // Захват мьютекса
        omp_mutex_lock(h);
    }
}
```

Листинг 4.2

```
// Методы, которыми мы опирируем при работе с queue
queue(int) q;
push();
pop();
front();
...

t = 0;
mtx.lock();
q.push(1);
cv.notify();
mtx.unlock();
...

mxt.lock();
while(q.empty())
    cv.wait(mtx);
int v = q.front();
q.pop();
mtx.unlock();

cout << v;
```

Рекомендации по работе с мьютексами

1. Используйте мьютексы для защиты совместно используемых данных.
2. Убедитесь, что каждый вызов `lock()` сопровождается вызовом `unlock()`.
3. Проверяйте возможные ошибки, связанные с покинутыми мьютексами.
4. Рассматривайте альтернативы (например, спин-блокировки) для более эффективного выполнения в некоторых сценариях.

Листинг 4.3

```
#include <iostream>
#include <cstring>
#include <vector>
#include <iomanip>
#include <omp.h>

unsigned r_r(unsigned *V, unsigned n){
    unsigned sum = 0;
    int T;

    #pragma omp parallel
    {
        unsigned mysum;

        unsigned t = omp_get_thread_num();
        unsigned T = omp_get_num_threads();

        unsigned s = n / T;
        unsigned b = n % T;

        if (t < b)
            b = ++s * t;
```

```
        else
            b += s*t;

        for (int i = b; i < e; i++)
            mysum += V[i];

        # pragma omp critical
        {
            sum += mysum;
        }
    }

    std::cin.get();
    return sum;
}
```